Vysoké učení technické v Brně
Fakulta informačních technologií

Dokumentácia projektu na predmety IFJ a IAL

Implementácia prekladača imperatívneho jazyka IFJ17

Tím 126, varianta II

Autori:

Ján Pristaš, xprist06 : 35% (vedúci)

Mária Masárová, xmasar13 : 35%

Roman Yaremchuk, xyarem00 : 30%

David Kollár, xkolla07 : 0%

06.12.2017

1 Úvod

Táto práca vznikla ako dokumentácia projektu do predmetu Formálne jazyky a prekladače. Projekt nám pomohol využiť naše teoretické a praktické vedomosti, ktoré sme nazbierali na prednáškach a pomocou nich vytvoriť dielo, ktoré je využiteľné vo svete programovania. Táto dokumentácia podrobne popisuje tvorbu prekladača IFJ17, prácu v tíme, ako aj implementáciu jednotlivých algoritmov a modelov, ktoré boli použité.

2 Práca v tíme

2.1 Rozdelenie úloh a komunikácia

Na začiatku semestra, prebehlo prvé stretnutie, kde sme sa zoznámili s projektom a rozdelili si prácu medzi jednotlivých členov. Bolo treba urobiť lexikálny analyzátor, syntaktický analyzátor ako aj sémantické kontroly, precedenčnú analýzu a inštrukcie. Roman dostal za úlohu urobiť scanner. Mária a Ján mali urobiť parser s precedenčnou analýzou a sémantickými kontrolami a David mal mať na starosti inštrukcie. S komunikáciou v tíme nastali problémy hneď na začiatku. Po prvom stretnutí nasledovala dlhá pauza, kedy sme sa vôbec ako celý tím nestretli. Samozrejme, očakávalo sa, že každý člen svoju prácu robí aj tak. Asi týždeň pred prvým pokusným odovzdaním, sme mali hotový lexikálny analyzátor, syntaktický analyzátor s precedenčnou analýzou a časť sémantiky. Komunikácia v tíme však stále zaostávala. Po prvom pokusnom odovzdaní sme zistili, že na našom projekte treba ešte veľa dorobiť. Bohužial sme zistili, že na projekte robíme iba dvaja. Vedúci napísal správu, kde sa opýtal, či si všetci myslia, že si zaslúžia 25% hodnotenia. Odvtedy sa nám už jeden člen vôbec neozval. Tretí člen sa po správe začal oveľa aktívnejšie zapájať do projektu. Podľa týchto skúseností sme urobili aj rozdelenie bodového hodnotenia.

2.2 Verzovací systém

Keďže práca na tomto projekte vyžadovala veľké množstvo fixovania a upravovania súborou, rozhodli sme sa použiť verzovací systém GIT. Vďaka nemu sme mali neustále prehľad o aktuálnej verzii nášeho programu ako aj o množstve práce, ktorú vykonali jednotlivý členovia.

2.3 Kontrola testovania

Ako som už spomínala aktívne sme sa podielali na projekte celý čas dvaja. Spolu sme konzultovali všetky zmeny a kontrolovali už implementované časti kódu. Prešli sme vždy celý kód a opravili chyby, na ktoré sme narazili. Osvečilo sa nám párové programovanie, kedy jeden programoval a druhý kontroloval. Bolo to rýchle, pretože kým jeden písal kód, druhý mohol hneď odhaliť, či nerobí nejakú chybu. Tento istý spôsob sme využili aj pri testovaní nášho programu. Keď sme spustili testy a projekt vyhodil nejaké chyby, väčšinou lexikálne a sémantické riešil jeden, zatiaľ čo ďalší sa sústredil na syntax. Testy sme spúštali celý čas. Po každej časti sa urobili testy, ktorými sme overili jej validitu. Veľa krát sa nám však stalo, že ak sme aj jednu časť vyhlásili za validnú, po pridaní ďalšej časti sme narazili ešte na množstvo neodhalených problémov.

3 Implementácia

3.1 Hašovacia tabuľka

Hašovacia tabuľka je štruktúra, ktorá asociuje kľúče s hodnotami a v tomto projekte bola využitá pre našu tabuľku symbolov. Tabuľka symbolov je abstraktná datová štruktúra, ktorá sa využíva pre uloženie všetkých identifikátorov, ktoré prekladač nájde v zdrojovom kóde programu. Implementácia hašovacej tabuľky bola z väčšej časti prebraná z domácej úlohy č.2 na predmete IAL. V projekte využívame 2 typy hašovacej tabuľky. Globálnu tabuľku, ktorá obsahuje všetky funkcie a lokálnu tabuľku, ktorú má každá funkcia vlastnú. Pôvodne sme tabuľku implementovali iba jednu, avšak po zvážení sme počet zvýšili na dve, pretože sa nám nedarilo v jednej tabuľke ukladať premenné, ktoré majú v rôznych funkciách rovnaké meno.

3.2 Lexikálna analýza

Lexikálna analýza je implementovaná v súbore `scanner.c`. Lexikálnu analýzu, ktorú možeme nazvať aj scanner sme implementovali ako konečný automat. Diagram konečného automatu môžete vidieť na strane 4 v Prílohach. Scanner, na pokyn syntaktického analyzátora, postupne prechádza vstupný súbor v jazyku IFJ17. Podľa znakov, ktoré mu prichádzajú na vstupe, lexikálny analyzátor vie, o aký token sa jedná. Tento token je reprezentovaný ako štruktúra, ktorá obsahuje typ tokenu a v niektorých prípadoch aj iné možnosti. Typ tokenu je vo forme číselnej konštanty, ktorá určuje o aký typ sa jedná. Po načítaní bieleho znaku môže lexikálny analyzátor poslať syntaktickému analyzátoru vyhodnotený token.

V prípade, že scanner z vstupného súboru načíta znak, ktorý nepozná a ktorý nezapadá do konečného automatu, ukončí program lexikálnou chybou.

Pri premenných a názvoch funkcií sa do štruktúry tokenu, konkrétne do premennej `meno`, uloží názov premennej alebo funkcie. Do tejto premennej sa taktiež ukladajú kľúčové slová. Scanner vie taktiež vyhodnocovať escape sekvencie.

3.3 Syntaktická analýza

Syntaktická analýza je implementovaná v súbore `parser.c`. Môžeme ju nazvať taktiež syntaktický analyzátor alebo parser. Syntaktická analýza je analýza zhora dolu a je implementovaná pomocou LL gramatiky. LL gramatiku môžete vidieť v Prílohach na strane 5 a 6. Gramatika popisuje povolenú štruktúru vstupného kódu.

Pomocou funkcie `GetToken()` parser predá riadenie scanneru, ktorý načíta vstupné znaky a vráti mu výsledný token. Parser skontroluje, či daný token zodpovedá tokenu, ktorý očakával. V prípade, že príde token, ktorý nezapadá do LL gramatiky, parser ukončí program so syntaktickou chybou.

Syntaktická analýza používa na spracovanie výrazov precedenčnú analýzu. V prípade, že parser narazí na výraz, predá riadenie precedenčnej analýze zavolaním funkcie `prec()`.

3.3.1 Precedenčná analýza

Precedenčná analýza je implementovaná v súbore `precedence.c`. Precedenčná analýza vycháza z precedenčej tabuľky, ktorá definuje pravidlá pre postupnosť operácií. Používa šiftovanie a redukovanie na to, aby zo zložitého výrazu dostala E. V prípade, že sa jej to podarí, je daný výraz vyhodnotený

ako správny. V opačnom prípade, program končí so syntaktickou chybou. Precedenčnú tabuľku môžete vidieť v Prílohách na strane 7.

Aby mohla precedenčná analýza výraz považovať za správny, musí prejsť množstvom krokov. Prvý krok, spočíva v pridaní znaku dolár na zásobník. Nasleduje porovnávanie výrazu s vrcholom zásobníka. Podľa precedenčnej tabuľky program vie, či treba znak výrazu pridať na zásobník alebo musí redukovať znaky v zásobníku. Na pridávanie slúži funkcia shift() a na redukovanie funkcia `reduce()`. Tento cyklus sa opakuje, kým nie je výraz zredukovaný na E alebo v prípade neúspešného zredukovania skončí program so syntaktickou chybou.

V prípade, že analýza narazí na volanie funkcie, predá riadenie naspäť parseru, ktorý kontroluje správnosť volania funkcie.

3.4 Kontrola sémantiky

Kontrola sémantiky prebieha počas syntaktickej a precedenčnej analýzy. Kontroluje, či zadaná premenná alebo funkcia je v tabuľke symbolov. Tento prípad vedie na sémantickú chybu číslo 3. K tejto chybe dochádza v prípade, že chceme použiť premennú alebo funkciu, ktorá nebola definovaná alebo aspoň deklarovaná. K tejto chybe taktiež dochádza, ak sa nezhoduje počet parametrov pri definovaní a deklarovaní funkcie. V prípade volania funkcie sa musí taktiež zhodovať počet parametrov.

K sémantickej chybe číslo 4 dochádza v prípade, keď sa snažíme do premennej nejakého typu priradiť premennú iného, nekompatibilného, typu. Môže ísť napríklad aj o návratovú hodnotu funkcie, keď je funkcia jedného typu ale vracia iný typ. Taktiež k tejto chybe dochádza v prípade, že sa parametre deklarovanej funkcie a parametre tej istej definovanej funkcie nezhodujú v type jednotlivých parametrov. Pri volaní funkcie je taktiež potrebné skontrolovať typ parametrov.

3.5 Inštrukcie

Ako posledné malo prísť vytváranie inštrukcií. Bohužiaľ, sa nám pre nezáujem štvrtého člena a pre nedostatok času, nepodarilo túto časť naimplementovať.

4 Záver

Tento projekt nám dal mnoho vedomostí a užitočných skúseností, ktoré v budúcnosti využijeme. Samozrejme, sme si pri práci na projekte prešli aj množstvom problémov. Prvým bol nedostatok času. Keďže predmet niektorí opakujeme, chceli sme mu venovať čo najväčšie úsilie. Zo začiatku sa nám to darilo, no akonáhle prišli projekty z tretiackych predmetov, museli sme čas rozdeliť aj pre ne. Ďalším problémom bola komunikácia a práceschopnosť niektorých členov.

Práve kvôli týmto problémom sa výsledok odzrkadlil na stave projektu.

Prílohy

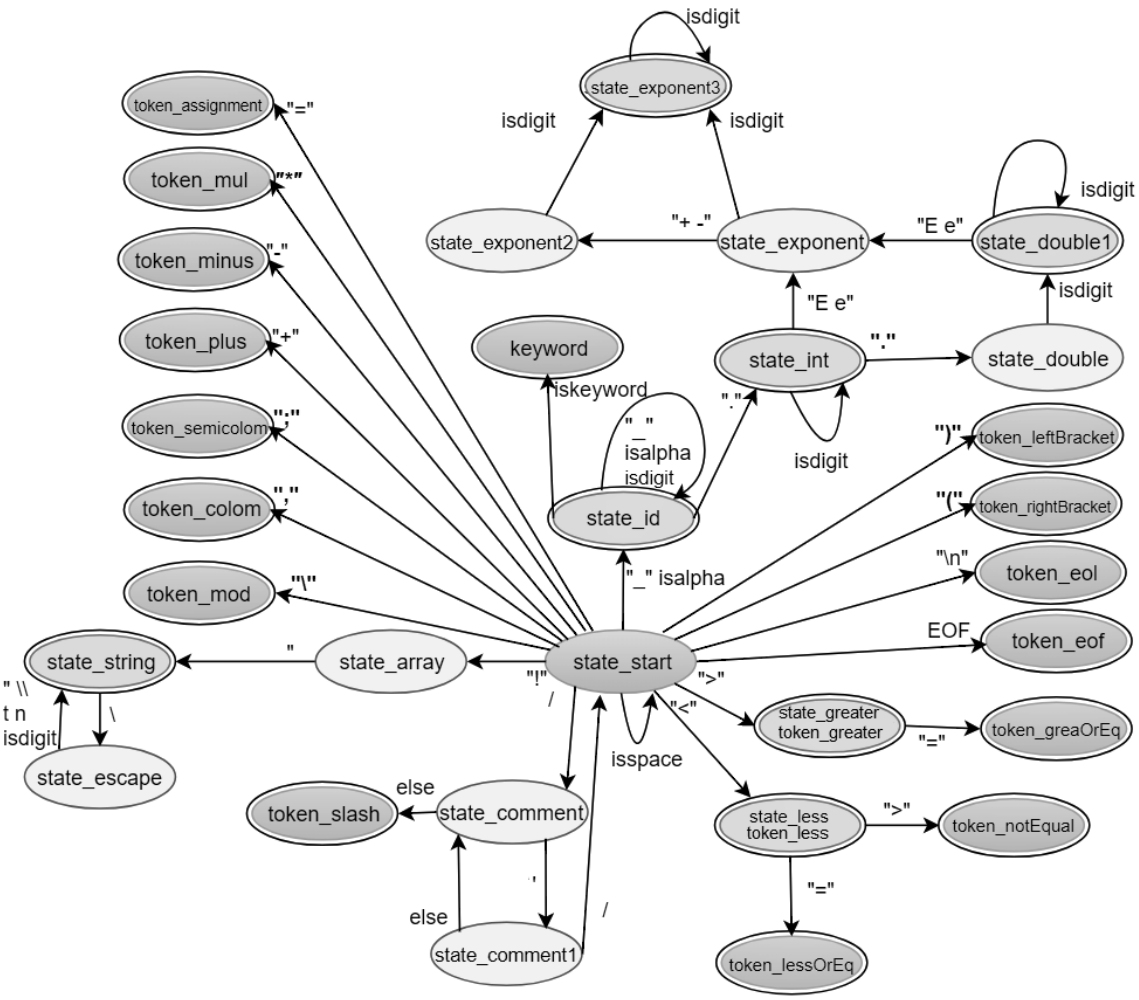

Diagram konečného automatu pre lexikálnu analýzu

```
// E - empty (nič tam už nie je)
<start>                 ->      <func> <scope>

// Hlavne telo programu
<scope>         ->      SCOPE EOL <scope_body> END SCOPE

<scope_body>->          E
<scope_body>->          DIM ID AS <type> <dim_n> <scope_body>
<scope_body>->          <body> <scope_body>

<cond_body>->           E
<cond_body>->           <body><cond_body>

<body>          ->      E
<body>          ->      EOL
<body>          ->      ID = <expr> EOL
<body>          ->      <condition>
<body>          ->      <loop>
<body>          ->      PRINT <printable>
<body>          ->      INPUT ID EOL

<dim_n>         ->      EOL
<dim_n>         ->      = <expr> EOL

<printable>     ->      <expr> ; <printable_n>

<printable_n> ->        <printable>
<printable_n> ->        EOL

<expr>          ->      val <operation>
<expr>          ->      ID <operation>
<expr>          ->      ID(<param>)

<b_expr>        ->      <expr> <b_op> <expr>
<b_expr>        ->      <expr>

<condition>     ->      IF <b_expr> THEN EOL <cond_body> ELSE EOL <cond_body> END IF EOL

<loop>          ->      DO WHILE <b_expr> EOL <cond_body> LOOP EOL

<param>         ->      E
<param>         ->      ID <params>

<params>        ->      E
<params>        ->      , ID <params>


// Deklaracia a definicia funkcii
<func>          ->      E
<func>          ->      <declare>
<func>          ->      <define>

<declare>       ->      DECLARE FUNCTION ID (<func_param>) AS <type> EOL <func>

<define>        ->      FUNCTION ID (<func_param>) AS <type> EOL <func_dim> END FUNCTION EOL<func>

<func_param>->          <first_func_p>
<func_param>->          E
```

Prvá časť LL gramatiky pre syntaktickú analýzu

```
<first_func_p>->      ID AS <type> <func_params>

<func_params>->       , ID AS <type> <func_params>
<func_params>->       E

<func_dim>    ->      E
<func_dim>    ->      DIM ID AS <type> <dim_n> <func_dim>
<func_dim>    ->      <func_body> <func_dim>

<func_body>   ->      E
<func_body>   ->      EOL
<func_body>   ->      ID = <expr> EOL
<func_body>   ->      <func_condition>
<func_body>    ->     <loop>
<func_body>   ->      PRINT <printable>
<func_body>   ->      INPUT ID EOL
<func_body>   ->      RETURN <expr> EOL

<f_cond_body>->       E
<f_cond_body>->       <func_body><f_cond_body>

<func_condition>->    IF <b_expr> THEN EOL <f_cond_body> ELSE EOL <f_cond_body> END IF EOL


<type>        ->      INTEGER
<type>        ->      STRING
<type>        ->      DOUBLE

<operation>   ->      <op> <expr>
<operation>   ->      E

<op>          ->      +
<op>          ->      -
<op>          ->      *
<op>          ->      /
<op>          ->      \

<b_op>        ->      <
<b_op>        ->      >
<b_op>        ->      <=
<b_op>        ->      >=
<b_op>        ->      =
<b_op>        ->      <>
```

Druhá časť LL gramatiky pre syntaktickú analýzu

	+	-	*	/	\	<	>	=	<=	>=	<>	(	)	i	$
+	>	>	<	<	<	>	>	>	>	>	>	<	>	<	>
-	>	>	<	<	<	>	>	>	>	>	>	<	>	<	>
*	>	>	>	>	>	>	>	>	>	>	>	<	>	<	>
/	>	>	>	>	>	>	>	>	>	>	>	<	>	<	>
\	>	>	<	<	>	>	>	>	>	>	>	<	>	<	>
<	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
>	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
=	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
<=	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
>=	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
<>	<	<	<	<	<	X	X	X	X	X	X	<	>	<	>
(	<	<	<	<	<	<	<	<	<	<	<	<	=	<	X
)	>	>	>	>	>	>	>	>	>	>	>	X	>	X	>
i	>	>	>	>	>	>	>	>	>	>	>	X	>	X	>
$	<	<	<	<	<	<	<	<	<	<	<	<	X	<	X

1. E -> E + E
2. E -> E - E
3. E -> E * E
4. E -> E / E
5. E -> E \ E
6. E -> E < E
7. E -> E > E
8. E -> E = E
9. E -> E <> E
10. E -> E <= E
11. E -> E >= E
12. E -> (E)
13. E -> i

Tabuľka pre precedenčnú syntaktickú analýzu